PARSING IN TAMIL: PRESENT STATE OF ART

S.Rajendran
rajushush@gmail.com, 2013

Parsing is actually related to the automatic analysis of texts according to a grammar. Technically, it is used to refer to practice of assigning syntactic structure to a text. It is usually performed after basic morphosyntactic categories have been identified in a text. Based on different grammars parsing brings these morphosyntactic categories into higher level syntactic relationships with one another. The survey of the state of art of parsing in Tamil reflects upon the global scenario. More or less the trends of the global arena in natural language processing are very much represented in Tamil too.

Over view of the global scenario

We try to understand larger textual units by combining our understanding of smaller ones. The linguistic theory aims to show how these larger units of meaning arise out of the combination of the smaller ones. This is modeled by means of a grammar. Computational linguistics then tries to implement this process in an efficient way. Traditionally the task is subdivide into syntax and semantics; syntax describes how the different formal elements of a textual unit, most often the sentence, can be combined; semantics describes how the interpretation is calculated. In most language technology applications the encoded linguistic knowledge, i.e., the grammar, is separated from the processing components. The grammar consists of a lexicon, and rules that syntactically and semantically combine words and phrases into larger phrases and sentences. A variety of representation languages have been developed for the encoding of linguistic knowledge. Some of these languages are more geared towards conformity with formal linguistic theories, others are designed to facilitate certain processing models or specialized applications. Several language technology products on the market today employ annotated phrase-structure grammars, grammars with several hundreds or thousands of rules describing different phrase types. Each of these rules is annotated by features, and sometimes also by expressions, in a programming language. When such grammars reach a certain size they become difficult to maintain, to extend, and to reuse. The resulting systems might be sufficiently efficient for some applications but they lack the speed of processing needed for interactive systems (such as applications involving spoken input) or systems that have to process large volumes of texts (as in machine translation).

In current research, a certain polarization has taken place. Very simple grammar models are employed, e.g., different kinds of finite-state grammars that support highly efficient processing. Some approaches do away with grammars altogether and use statistical methods to find basic linguistic patterns. On the

other end of the scale, we find a variety of powerful linguistically sophisticated representation formalisms that facilitate grammar engineering. The most prevalent family of grammar formalisms currently used in computational linguistics is constraint based.

Morphological Analysis in Tamil

Tamil is a Dravidian language. It is a verb final, relatively free-word order and morphologically rich language. Like other Dravidian languages, Tamil is agglutinative. Computationally, each root word can take a few thousand inflected word-forms, out of which only a few hundred will exist in a typical corpus. Subject-verb argument is required for the grammaticality of a Tamil sentence. Tamil allows subject and object drop as well as verb less sentences. In addition, the subject of a sentence or a clause can be a possessive Noun Phrase (NP) or an NP in nominative or dative case. As Tamil is an agglutinative language, each root word can combine with multiple morphemes to generate word forms. For the purpose of analysis of such inflectionally rich languages, the root and the morphemes of each word has to be identified.

The global scenario has influenced the morphological analysis of Tamil. In the last decade, computational morphology has advanced further towards real-life applications than most other subfields of natural language processing. To build a syntactic representation of the input sentence, a parser must map each word in the text to some canonical representation and recognize its morphological properties. The combination of a surface form and its analysis as a canonical form and inflection is called a lemma. The main problems are:

1. morphological alternations: the same morpheme may be realized in different ways depending on the context.

2. morphotactics: stems, affixes, and parts of compounds do not combine freely, a morphological analyzer needs to know what arrangements are valid.

A popular approach to 1 is the cut-and-paste method. The canonical form is derived by removing and adding letters to the end of a string. The use of finite-state technology for automatic recognition and generation of word forms was introduced in the early 1980s. It is based on the observation that rules for morphological alternations can be implemented by finite-state transducers. It was also widely recognized that possible combinations of stems and affixes can be encoded as a finite-state network. An automaton containing inflected word forms can be upgraded to a morphological analyzer, for example, by adding a code to the end of the inflected form that triggers some predefined cut-and-paste operation to produce the lemma. Instead of cutting and pasting it at runtime, the entire lemma can be computed in advance and stored as a finite-state transducer

whose arcs are labeled by a pair of forms. The transducer format has the advantage that it can be used for generation as well as analysis. The number of nodes in this type of network is small, but the number of arc-label pairs is very large as there is one symbol for each morpheme-allomorph pair. A more optimal lexical transducer can be developed by constructing a finite-state network of lexical forms, augmented with inflectional tags, and composing it with a set of rule transducers. Lexical transducers can be constructed from descriptions containing any number of levels. This facilitates the description of phenomena that are difficult to describe within the constraints of the two-level model. Because lexical transducers are bidirectional, they are generally non-deterministic in both directions. If a system is only to be used for analysis, a simple finite-state network derived just for that purpose may be faster to operate.

The following is the list of computational morphological analysis attempted and/or implemented for Tamil:

1. Rajendran's Morphological Analyzer for Tamil: The first step towards a preparation of morphological analyzer for Tamil was initiated by *anusaraka* group of researchers under whose guidance Rajendran, Tamil University prepared a morphological analyzer for Tamil for Translating Tamil into Hindi at the word level.

2. Genesan's Morphological Analyzer for Tamil: Ganesan developed a morphological analyzer for Tamil to analyze CIIL corpus. He exploits phonological and morphophonemic rules and takes into account morphotactic constraints of Tamil in building morphological analyzer for Tamil. Recently he has built an improved and efficient morphological parser.

3. Kapilan's Morphological Analyzer for Tamil Verbal Forms: Kapilan prepared a morphological analyzer for verbal forms in Tamil.

4. Deivasundaram's Morphological parser: Deivasundarm has prepared a morphological analyzer for Tamil for his Tamil Word Processor. He too makes use of phonological and morphophonemic rules and morphotnatic constraints for developing his parser.

5. AUKBC Morphological Parser for Tamil: AUKBC NLP team under the supervision of Rajendran prepared a Morphological parser for Tamil. The API Processor of AUKBC makes use of the finite state machinery like PCKimmo. It parses, but does not generate.

6. Vishnavi's Morphological Generator for Tamil: Vaishnavi researched for her M.Phil. dissertation on morphological generator for Tamil. The Vaishanvi's morphological generator implements the item and process model of linguistic description. The generator works by the synthesis method of PCKimmo.

7. Ramasamy's Morphological Generator for Tamil: Ramasamy has prepared a morphological generator for Tamil for MPhil dissertation.

8. Winston Cruz's Parsing and Generation of Tamil Verbs: Winston Cruz makes use of GSmorph method for parsing Tamil verbs. GSmorph too does morphotactics by indexing. The algorithm simply looks up two files to see if the indices match or not. The processor generates as many forms as it parses and uses only two files.

9. Vishnavi's Morphological Analyzer for Tamil: Vaishnavi again researched for her Ph.D. dissertation on the preparation of Morphological Analyzer for Tamil. She proposes a hybrid model for Tamil. It finds its theoretical basis in a blend of IA and IP models of morphology. It constitutes an in-built lexicon and involves a decomposition of words in terms of morphemes within the model to realize surface well-formed words-forms. The functioning can be described as defining a transformation depending on the morphemic nature of the word stem. The analysis involves a scanning of the string from the right to left periphery scanning each suffix at a time stripping it, and reconstructing the rest of the word with the aid of phonological and mophophonemic ruels exemplified in each instance. This goes on till the string is exhausted. For the sake of comparison she implements AMPLE and KIMMO models. She also evaluates TAGTAMIL, API Analyzer, and GSMorph. She concludes that Hybrid model is more efficient that the rest of the models.

10. Dhurai Pandi's Morphological Generator and Parsing Engine for Tamil Verb Forms: It is a full-fledged morphological generator and a parsing engine on verb patterns in modern Tamil.

11. RCILTS-T's Morphological analyzer for Tamil: Resource Centre for Indian Language Technological Solutions-Tamil has prepared a morphological analyzer for Tamil. It is named as atcharam. Atcharam takes a derived word as input and separate into root word and associated morphemes. It uses a dictionary of 20000 root words based on fifteen categories. It has two modules - noun and verb analyzer based on 125 rules. It uses heuristic rules to deal with ambiguities. It can handle verb and noun inflections.

12. RCILTS-T's Morphological generator for Tamil: Resource Centre for Indian Language Technological Solutions-Tamil has prepared a morphological generator also for Tamil. It is named as atchayam. Atchayam generates words when Tamil morphs are given as input. It has two major modules – noun and verb generators. The noun section handles suffixes like plural markers, oblique form, case markers and postpositions. The verb section takes tense and PNG makers, relative and verbal participle suffixes, and auxiliary verbs. It uses sandhi rules and125 morphological rules. It handles adjectives and adverbs. It has word and sentence generator interfaces.

Morphological Disambiguation in Tamil

Word-forms are often ambiguous. Alternate analyses occur because of categorial homonymy, accidental clashes created by morphological alternations, multiple functions of affixes, or uncertainty about suffix and word boundaries. The sentential context normally decides which analysis is appropriate. This is called disambiguation. There are two basic approaches to disambiguation: rule-based and probabilistic. Rule-based taggers typically leave some of the ambiguities unresolved but make very few errors; statistical taggers generally provide a fully disambiguated output but they have a higher error rate. Probabilistic (stochastic) methods for morphological disambiguation have been dominant since the early 1980s. Standard statistical methods can be applied to provide a fully disambiguated output.

Baskaran and Vijay-Shankar who have studies 'Influence of Morphology in Word Sense Disambiguation for Tamil' concludes in the following fashion: "The experiments conducted using both supervised and semi-supervised approaches clearly indicate that morphological inflections indeed affect the system performance, thus strongly suggesting need for morphology in the sense disambiguation of Tamil in particular and other inflectional languages in general."

Shallow Parsing in Tamil

We use the term shallow syntax as a generic term for analyses that are less complete than the output from a conventional parser. The output from a shallow analysis is not a phrase-structure tree. A shallow analyzer may identify some phrasal constituents, such as noun phrases, without indicating their internal structure and their function in the sentence. Another type of shallow analysis identifies the functional role of some of the words, such as the main verb, and its direct arguments. Systems for shallow parsing normally work on top of morphological analysis and disambiguation. The basic purpose is to infer as much syntactic structure as possible from the lemmata, morphological information, and word order configuration at hand. Typically, shallow parsing aims at detecting phrases and basic head/modifier relations. A shared concern of many shallow parsers is the application to large text corpora. Frequently partial analyses are allowed if the parser is not potent enough to resolve all problems. Church (1988) has designed a stochastic program for locating simple noun phrases which are identified by inserting appropriate brackets, [...].

Abney (1991) is credited with being the first to argue for the relevance of shallow parsing, both from the point of view of psycholinguistic evidence and from the point of view of practical applications. His own approach used hand-crafted cascaded Finite State Transducers to get at a shallow parse. Typical modules within a shallow parser architecture include the following:

1. Part-of-Speech Tagging. Given a word and its context, decide what the correct morphosyntactic class of that word is (noun, verb, etc.). POS tagging is a well-understood problem in NLP, to which machine learning approaches are routinely applied.

2. Chunking. Given the words and their morphosyntactic class, decide which words can be grouped as chunks (noun phrases, verb phrases, complete clauses, etc.)

3. Relation Finding. Given the chunks in a sentence, decide which relations they have with the main verb (subject, object, location, etc.).

Because shallow parsers have to deal with natural languages in their entirety, they are large, and frequently contain thousands of rules (or rule analogues). These rule sets also tend to be largely `soft', in that exceptions abound. Building shallow parsers is therefore a labour-intensive task. Unsurprisingly, shallow parsers are usually automatically built, using techniques originating within the machine learning (or statistical) community.

Parts of speech Tagging in Tamil

Parts of speech tagging scheme tags a word with its parts of speech in a sentence. It is done in three stages: pre-editing, automatic tag assignment, and manual post-editing. In pre-editing, corpus is converted to a suitable format to assign a part of speech tag to each word or word combination. Because of orthogrpahic similarity one word may have several possible POS tags. After initial assignment of possible POS, words are manually corrected to disambiguate words in texts.

1. Vasu Ranganathan's Tagtamil: Tagtamil by Vasu Ranganathan is based on Lexical phonological approach. Tagtamil does morphotactics of morphological processing of verbs by using index method. Tagtamil does both tagging and generation.

2. Ganesan's POS tagger: Ganesan has prepared a POS tagger for Tamil. His tagger works well in CIIL Corpus. Its efficiency in other corpora has to be tested. He has a rich tagset for Tamil. He tagged a portion of CIIL corpus by using a dictionary as well as a morphological analyzer. He corrected it manually and trained the rest of the corpus with it. The taggs are added morpheme by morpheme.

vandtavan: va_IV_ ndt_PT_avan_3PMS
pukkaLai : puu_N_PL_AC

3. kathambam of RCILTS-Tamil: Kathambam attaches parts of speech tags to the words of a given Tamil document. It uses heuristic rules based on Tamil linguistics for tagging and does not use either the dictionary or the morphological

analyzer. It gives 80% efficiency for large documents. It uses 12 heuristic rules. It identifies the tags based on PNG, tense and case markers. Standalone words are checked with the lists stored in the tagger. It uses 'Fill in rule' to tag 'unknown words. It uses bigram and identifies the unknown word using the previous word category.

Chunking in Tamil

Basically a chunker divides a sentence into its major-non-overlapping phrases and attaches a label to each. Chunker differ in terms of their precise output and the way in which a chunk is defined. Many do more than just simple chunking. Others just find NPs. Chunking falls between tagging (which is feasible but sometimes of limited use) and full parsing (which more useful but is difficult on unrestricted text and may result in massive ambiguity. The structure of individual chunks is fairly easy to describe, while relations between chunks are harder and more dependent on individual lexical properties. So chunking is a compromise between the currently available and the ideal processing output. Chunkers tokenise and tag the sentence. Most chunkers simply use the information in tags, but others look at actual words.

Noun Phrase Chunking in Tamil

Noun phrase chunking deals with extracting the noun phrases from a sentence. While NP chunking is much simpler than parsing, it is still a challenging task to build an accurate and very efficient NP chunker. The importance of NP chunking derives from the fact that it is used in many applications.

Noun phrases can be used as a pre-processing tool before parsing the text. Due to the high ambiguity of the natural language exact parsing of the text may become very complex. In these cases chunking can be used as a pre-processing tool to partially resolve these ambiguities. Noun phrases can be used in Information Retrieval systems. In this application the chunking can be used to retrieve the data's from the documents depending on the chunks rather than the words. In particular nouns and noun phrases are more useful for retrieval and extraction purposes. Most of the recent work on machine translation use texts in two languages (parallel corpora) to derive useful transfer patterns. Noun phrases also have applications in aligning of text in parallel corpora. The sentences in the parallel corpora can be aligned by using the chunk information and by relating the chunks in the source and the target language. This can be done lot more easily than doing word alignment between the texts of the two languages. Further noun phrases that are chunked can also be used in other applications where in depth parsing of the data is not necessary.

1. AUKBCRC's Noun Phrase Chunker for Tamil : The approach is a rule based one. In this method initially a corpus is taken and it is divided into two or more sets. One of these divided sets is used as the training data. The training data set is taken and manually chunked for noun phrases, thus evolving rules that can be applied to separate the noun phrases in a sentence. These rules serve as the base for chunking. The chunker program uses these rules and chunks the test data. The coverage of these rules is tested with this test data set. Precision and recall are calculated for this and the result is analyzed to check, if more rules are needed to improve the coverage of the system. If more rules are needed then additional rules are added and the same process as mentioned above is repeated to check for increase in the precision and recall of the system. The system is then tested for various other applications.

2. vaanavil of RCILTS-Tamil: vaanavil identifies the syntactic constituents of a Tamil sentence. It ouputs the parsed tree in a list form. It tackles both simple and complex sentences. Simple sentences can have a verb, many noun phrase, simple adverbs and adjectives. Complex sentences can have multiple adjectival, adverbial and noun clausal forms. In the case of sentences with multiple clauses, vaanavil syntactically groups the clauses based on the cue words and phrases. It makes of phrase structure grammar. It uses look-ahead to handle free word order. It handles ambiguity using 15 heuristic rules. It uses the morphological analyzer to obtain the root word.

Grammar Formalisms and syntactic parsing in Tamil

For processing a natural language certain formalisms are required. The grammatical models proposed by linguists, otherwise called as grammatical formalism try to capture the phonological, grammatical and semantic organization of natural language partially or fully. Grammatical formalisms are written with the purpose of comprehending the units and patterns found in all the levels of language. Computer scientists take the grammatical formalisms, modify them suitably for creating data-base procedures for machines so to make the machines process, recognize and produce natural language units and structures. Such a computational description is called as computational formalism.

The free word order feature of Tamil makes parsing a challenging task. There is a need to associate and link components that are not always adjacent to each other. A number of fomalisms have been made use for parsing in Tamil The widely used one is Context Free Grammar formalism coupled with finite state automata. Phrase Structure grammars have been designed on fixed word order languages like English. Tamil is a variable word order language. In a sentence features of words or grammatical constituents can be tightly coupled or loosely coupled. In fixed word order languages features like number and gender in the case of nouns and tense and number in the case of verbs are tightly coupled attachments to the respective syntactic category. The other linkages are loosely coupled and indicated by word proximity. In Tamil in addition to features

like number, gender and tense, case attachments of nouns and aspect and mood of verbs are tightly coupled through inflectional attachments and do not need word proximity to indicate dependency. So Tamil requires a different kind of grammatical formalism which rely on dependency rather than proximity. Tamil rely more on morphology than syntax in indicating grammatical functions.

In the global arena a very advanced and wide-spread class of linguistic formalisms are the so-called constraint-based grammar formalisms which are also often subsumed under the term unification grammars. They go beyond many earlier representation languages in that they have a clean denotational semantics that permits the encoding of grammatical knowledge independent from any specific processing algorithm. These formalisms are currently used in a large number of systems. Among the most used, constraint-based grammar models are Functional Unification Grammar (FUG), (Kay, 1984) Head-Driven Phrase-Structure Grammar (HPSG), (Pollard & Sag, 1994) Lexical Functional Grammar (LFG), (Bresnan, 1982), Categorial Unification Grammar (CUG), (Haddock, Klein, et al., 1987; Karttunen, 1989; Uszkoreit, 1986), and Tree Adjunction Grammar (TAG) (Joshi & Schabes, 1992). For these or similar grammar models, powerful formalisms have been designed and implemented that are usually employed for both grammar development and linguistic processing, e.g, LFG (Bresnan, 1982), PATR (Shieber, Uszkoreit, et al., 1983), ALE (Carpenter, 1992), STUF (Bouma, Koenig, et al., 1988), ALEP (Alshawi, Arnold, et al., 1991), CLE (Alshawi, 1992) TDL (Krieger & Schaefer, 1994)TFS (Emele & Zajac, 1990). Almost all ongoing European Union-funded language technology projects involving grammar development have adopted unification grammar formalisms.

1. Baskaran's Finite-state Machine for Syntactic Parsing: Finite-state Automata is one of the important techniques for parsing at all the level of a language structure. On experimental basis Baskaran (1984) has attempted a Finite-State-Machine for parsing sentences in Tamil.

2. Kumara Shanmugam's Syntactic Parser for Tamil: Keeping the characteristics of Tamil in mind Kumara Shanmugam (2004) has prepared a parser for Tamil. The parser he has designed carry out a complete morphological analysis of words of the sentences at the first level in order to help in dependency determination. The parser divides the sentences into two basic constituents noun part and verb part. In other words he has a one level syntax tree. Since Tamil has variable word order it is possible that the noun or verb parts could be discontinuous. Thus the parser uses the morphological analyzer to determine tightly coupled features which help in classification of the words. Unclassified words are classified based on heuristics. Dependencies between noun head and verb head and their respective modifiers are tackled with the help of dependency rules. Sentence patterns are then used to analyze the sentence. The selection of the sentence pattern depends on information provided by the morphological analyzer. The addition of rules for semantic dependencies can enhance the performance of the parser.

3. Shanmugam's Parsing Techniques: Shanmugam while proposing a program for syntactic parsing in Tamil makes the following comments: "Structural description of the units of a language can be provided by the grammar of language, making use of a principle called 'Projection Principle'. According to this principle, as in transformational grammatical treatise, the structure of a sentence or phrase can be projected or plotted from the lexical specification of the head of the phrase or sentence. That projected structure will be abstract structure which will be modified with due substitution of appropriate lexical items. Shanmugam (2002) advocates for minimalist program for Tamil parsing. All grammatical formalisms identify lexicon and certain procedures for creating and manipulating grammatical structures. Minimalist program which is a grammatical model and an extension of GB framework was proposed by Chomsky to expose the grammatical patterns found in languages. Some of his MPhil and Ph.D. students have worked for their dissertation on Context Free Grammar Formalism, Transformational Generative Grammar Formalism, Projection Principle, and Minimalist Program and prepared syntactic parser models for Tamil based on the formalism they have chosen.

4. RCILTS-Tamil syntactic parser: The parser handles simple and complex sentences with multiple nouns, adjective and adverb clauses. Handling of conjunction has been tackled to a limited extent. The addition of rules for semantic dependencies can enhance the performance of the parser. The seems to parse sentences in terms of clauses such as noun clauses, verb clauses, adjective clauses and adverbial clauses. The clauses have been parsed into categories.

Future Directions

The issue that dominates current work in parsing and language modeling is to design parsers and evaluation functions with high coverage and precision with respect to naturally occurring linguistic material (for example, news, stories, spontaneous speech interactions). Simple high-coverage methods such as n-gram models miss the higher-order regularities required for better prediction and reliable identification of meaningful relationships, while complex hand-built grammars often lack coverage of the tail of individually rare but collectively frequent sentence structures. Automated methods for grammar and evaluation function acquisition appear to be the only practical way to create accurate parsers with much better cover. The challenge is to discover how to use linguistic knowledge to constrain that acquisition process.

Arulmozhi, S. 1998. Aspect of Inflectional Morphology – A Computational Approach. Ph.D. dissertation, University of Hyderabad.

Anandan P, Rajani Parthasarathy, Geetha, T.V. 2001. "Morphological analyzer for Tamil". ICON 2002, RCILTS-Tamil Anna University, Chennai.

Anandan, P, Rajani Parthasarathy, Geetha, T.V. 2001. “Morphological Generator for Tamil” in Tamil Internet 2001 Conference Proceedings, Malaysia.

Brouchers on ‘Language Technology Products’ of the Resource Center for Indian Language Technology Solutions – Tamil, Chennai

Balakrishnan, R. 2002. Morphology and Tamil Computing. Paper read in International Seminar on Tamil Computing, February 27, 28, 2002, Madras University.

cevveel kapilan, 1994. KaNippoRi vazhi tamizh vanikaLin prauppaayvu. Chennai: puttaakka mozhiyial kazhakam.

cupasri, je. 2005. *tamizh vinaic coRkaLin urupaniayal aayvi. aayviyal niRainjar aayveeTu, mozhiyiyal tuRai, tamizhp palkalaikkazhakam, tanjaavuur.*

Deivasundaram, N. and Gopal, A. 2003. ‘Computational Morphology of Tamil’ In B. Ramakrishna Reddy (ed.) Word Structure in Dravidian, Kuppam: Dravidian University, 406-410.

Duraipandi, R. “The Mophological Generator and Parsing Engines of Tamil Verb forms”. In Tamil Internet 2006.Chennai: Asian Printers.

Ganesan, M. 1994. “Functions of Morphological Analyzer Developed at CIIL, Mysore” in Harikumar Basi (ed.) Automatic Translation (seminar proceedings), Thiruvanthapuram: ISDL

Ganesan, M. 2003. “Computational Morphology of Tamil”. In B. Ramakrishna Reddy (ed.) Word Structure in Dravidian, Kuppam: Dravidian University, 399-405.

Ganesan, M and Francis Ekka. 1994. Morphological Analyzer for Indian Languages. Agarwarl Pani (eds) Information Technology Applications in Language, Script and Speech. New Delhi:BPB Publication.

iraamacundtari, 2005. *tamizh peyarc coRkaLin urupaniayal aayvi. aayviyal niRainjar aayveeTu, mozhiyiyal tuRai, tamizhp palkalaikkazhakam, tanjaavuur*.

Kumara Shanmugam, B. 2004. Parse representation of Tamil syntax. MS Thesis, submitted to Anna University, Chennai.

Language Analysis and Understading. In Survey of the State of Art in Human Language Technology. (A downloaded script)

Rajendran S, Arulmozi S, Ramesh Kumar S, & Viswanathan S. 2003. Computational Morpohology of Verbal Complex In B. Ramakrishna Reddy (ed.) Word Structure in Dravidian, Kuppam: Dravidian University, 376-398.

Ramaswamy, V. 2000. Morphological Generator for Tamil. Unpublished M.Phil dissertation. University of Hyderabad.

Ranganathan, V. 1997. A Lexical Phonology Approach to Processing Tamil Word by Computer. International Journal of Dravidian Linguistics 26.1:

Shanmugam, C. 2001. “Computer Analysis of Simple Sentence in Tamil”. Paper read in UGC-SAP National Seminar on Computational Linguistics and Dravidian Languages, 22-24 February, 2001, CAS in Linguistics, Annamalai University, Annamalainagar.

---------2002. “Grammar and Parser: A Program for Syntactic Parsing in Tamil,” International Seminar on Tamil Computing, 27-28 February and March 1, 2002, University of Madras, Chennai.

----------“Minimalist Program for Tamil Parsing”.

Sivashanmugam, C. 2000. “A Model for Computer analysis of Verbs in Tamil”. in Working Papers in Linguistics, Department of Linguistics, Bharathiyar University, Coimbatore.

----------2002. Morphological Processor for Negative Constructions in Tamil”. Indian Conference on Natural Language Processing, Anna University, Chennai.

Vaishnavi Ramaswamy. 2000. A Morphological Generator for Tamil. M.Phil Dissertaion Subbmitted to University of Hyderabad.

Vaishnavi Ramaswamy. 2003. A Morphological Analyzer for Tamil. Ph.D Dissertaion Subbmitted to University of Hyderabad.

Viswanathan, S., Rameshkumar, S. Kumara Shanmugam, B. Arulmozhi. S. & Vijay Shnakar, K.2003. A Tamil Morphological Analyzer. In Rajeev Sangal, S.M. Bendre & Udaya Narayana Sigh (eds.), Recent Advances in Natural Language Processing. Mysore: CIIL

Winston Cruz, S. 2002. Parsing and Generation of Tamil Verbs in GSMorph. M.Phil. dissertation submitted to the University of Hyderabad.